

# 【完全理解】

# TCPアルゴリズム

## 基本動作編 《ロスベース方式》

- 確実にデータが届く仕組み
- ネットワーク輻輳を把握し効率的に伝送

# 全体目次

**1. TCP概要**

**2. TCPアルゴリズム**

**3. まとめ**

# 1. TCP概要

1-1. RFCとは？

1-2. TCP/IPとは？

1-3. プロトコルスタック

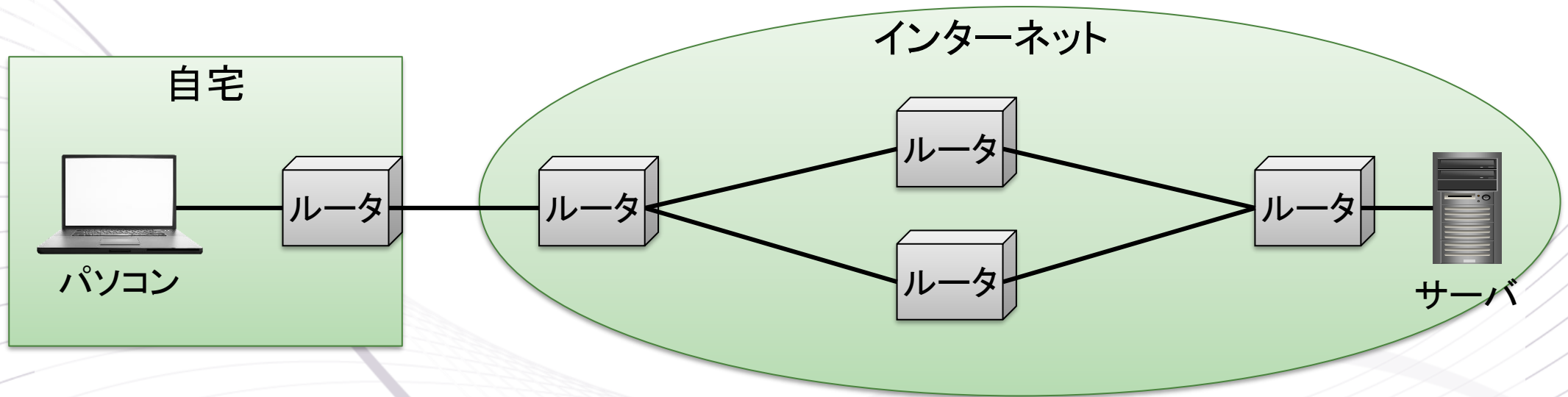
1-4. パケットデータ (キャプチャ)

1-5. TCPバージョン

1-6. TCPデータフォーマット

# 1-1. RFCとは？

インターネット通信に関わる全ての機器は予め規定された技術でデータを扱うことが必要。  
そのため、インターネット技術の標準化（取り決め）が行われている。  
この標準化をRFCという。

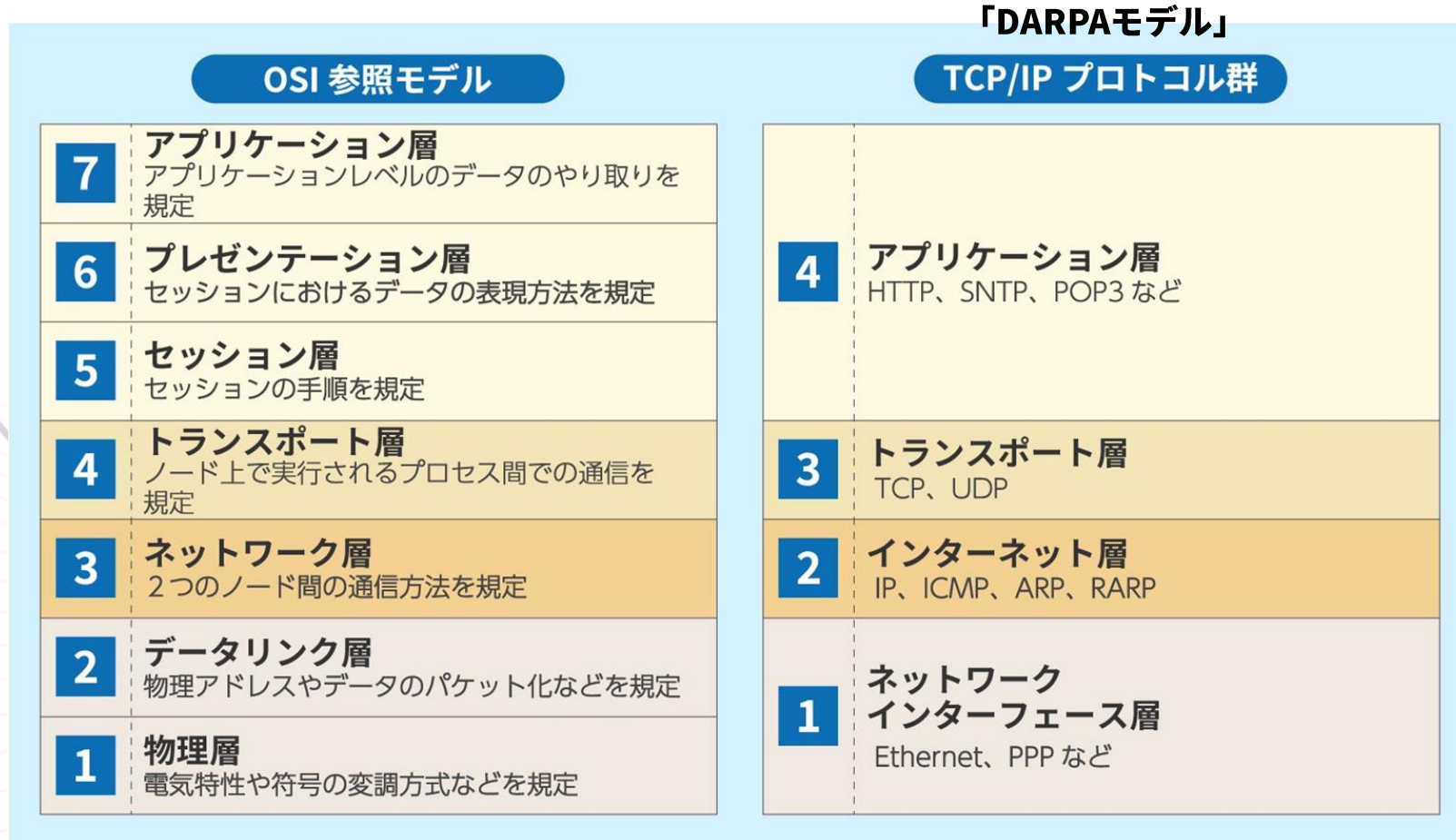


- 標準化を行う団体：IETF (Internet Engineering Task Force)
- 標準化された規定：RFC (Request for Comments)
  - 1) IP : RFC791
  - 2) TCP : RFC9293 2022年8月公開 (RFC793廃止)

## 1-2. TCP/IPとは？

- **TCP/IP** インターネットでの通信を行うための規定
  - ① **IP** : インターネット上の住所を表す役割
  - ② **TCP** : 通信路の状態に応じて効率的かつ確実にデータを届ける役割
- **UDP/IP**
  - ③ **UDP** : 届けるだけで通信制御をおこなわない。

# 1-3. プロトコルスタック



- 送受信端末で各層の利用する規定を一致させる
- 各層間の依存関係がない。  
各々の層毎に利用する規定を決める

# 1-4. パケットキャプチャ (Wireshark)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.7	157.7.107.210	TCP	66	53275 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000364	192.168.1.7	157.7.107.210	TCP	66	59662 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	0.015081	157.7.107.210	192.168.1.7	TCP	66	80 → 53275 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1414 SACK_PERM=1 WS=128
4	0.015193	192.168.1.7	157.7.107.210	TCP	54	53275 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
5	0.016168	192.168.1.7	157.7.107.210	HTTP	614	GET / HTTP/1.1
6	0.016390	157.7.107.210	192.168.1.7	TCP	66	80 → 59662 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1414 SACK_PERM=1 WS=128
7	0.016462	192.168.1.7	157.7.107.210	TCP	54	59662 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8	0.031443	157.7.107.210	192.168.1.7	TCP	60	80 → 53275 [ACK] Seq=1 Ack=561 Win=30336 Len=0
9	0.542488	157.7.107.210	192.168.1.7	TCP	392	80 → 53275 [PSH, ACK] Seq=1 Ack=561 Win=30336 Len=338 [TCP segment of a reassembled PDU]
10	0.543525	157.7.107.210	192.168.1.7	TCP	2882	80 → 53275 [ACK] Seq=339 Ack=561 Win=30336 Len=2828 [TCP segment of a reassembled PDU]
11	0.543622	192.168.1.7	157.7.107.210	TCP	54	53275 → 80 [ACK] Seq=561 Ack=3167 Win=131328 Len=0
12	0.544684	157.7.107.210	192.168.1.7	TCP	8538	80 → 53275 [ACK] Seq=3167 Ack=561 Win=30336 Len=8484 [TCP segment of a reassembled PDU]

> Frame 5: 614 bytes on wire (4912 bits), 614 bytes captured (4912 bits) on interface \Device\NPF\_{EE298557-9F26-440F-870F-072F5825E194}, id 0  
> Ethernet II, Src: Tp-LinkT 09:d6:7d (28:ee:52:09:d6:7d), Dst: Mitsubis 86:d6:65 (10:4b:46:86:d6:65)  
> Internet Protocol Version 4, Src: 192.168.1.7, Dst: 157.7.107.210  
> Transmission Control Protocol, Src Port: 53275, Dst Port: 80, Seq: 1, Ack: 1, Len: 560  
> Hypertext Transfer Protocol

Ethernet: 14Byte

IP: 20Byte

TCP: 20Byte

0000	10 4b 46 86 d6 65 28 ee 52 09 d6 7d 08 00 45 00	·KF·e(· R·}··E·
0010	02 58 15 30 40 00 80 06 18 e7 c0 a8 01 07 9d 07	·X·0@· ··· ······
0020	6b d2 d0 1b 00 50 38 57 b9 9e 16 b5 52 a1 50 18	k· ··P8W ····R·P·
0030	02 01 e5 e0 00 00 47 45 54 20 2f 20 48 54 54 50	· ·····GE T / HTTP
0040	2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 6d 61 6e 61	/1.1·Ho st: mana
0050	6b 61 6e 2e 6e 65 74 0d 0a 43 6f 6e 6e 65 63 74	kan.net· ·Connect
0060	69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d	ion: kee p-alive·
0070	0a 55 70 67 72 61 64 65 2d 49 6e 73 65 63 75 72	·Upgrade -Insecur
0080	65 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a 55	e-Request s: 1·U
0090	73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c	ser-Agen t: Mozil

# 1-5. TCPバージョン

今回の説明

- 初期標準の最もベーシックなTCP
  - 「Reno」 「NewReno」 \*1
- 高遅延・広帯域（ロングファットパイプ）向け
  - 「BIC」 「CUBIC」 「H-TCP」 \*1、 「Fast TCP」 \*2、 「Illinois」 \*3
- 低遅延・広帯域であるデータセンターに向け
  - 「DCTCP」 \*3
- モバイル回線やWi-Fiなど無線環境向け
  - 「Veno」 「Westwood」 \*2

## 《参考1：方式》

\*1：ロスベース方式

\*2：遅延ベース方式

\*3：上記2つのハイブリッド方式

## 《参考2：各OS実装》

Linux、Android：CUBIC

OS X：NewReno

Windows：CTCP/DCTCP



# 1-6. TCPデータフォーマット

- 規定 : RFC 9293

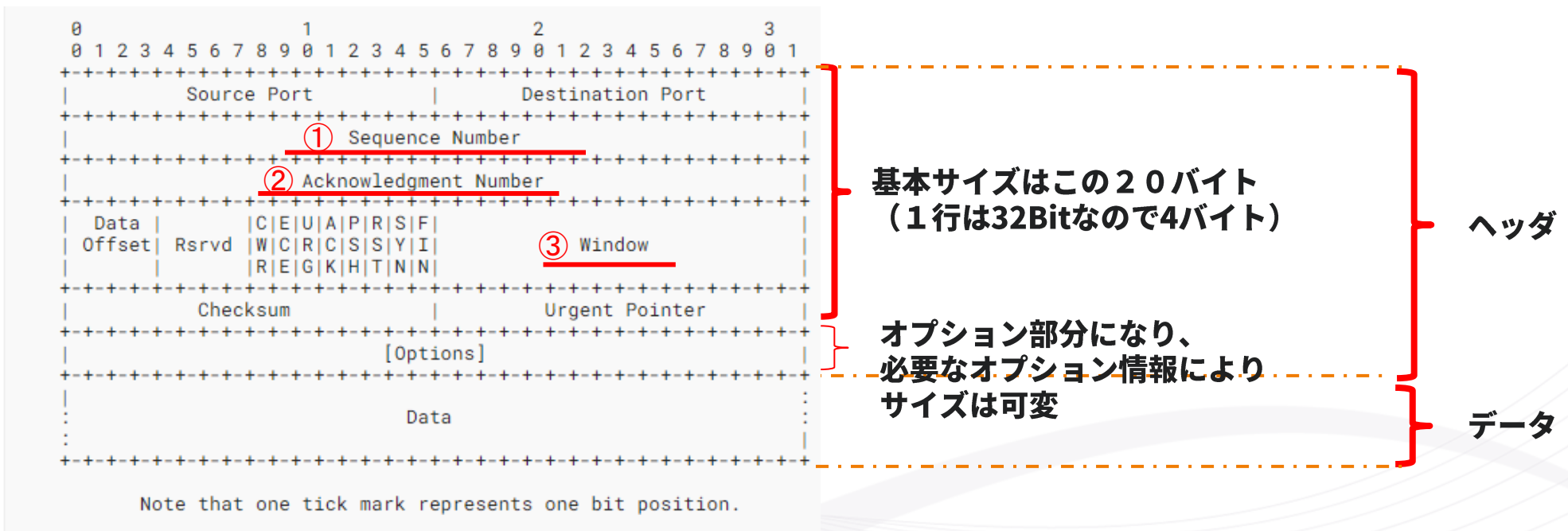


Figure 1: TCP Header Format

①	<b>Sequence Number</b> (シーケンスナンバー)	送信元端末が送信データに番号 (Sequence Number) をつけて送信します。
②	<b>Acknowledgment Number</b> (アックノレッジナンバー)	受信端末が届いた「Sequence Number」を元に次に受信を期待する「Sequence Number」を確認応答 (Acknowledgment Number) として送信元に通知します。
③	<b>Window</b> (ウィンドウ)	受信端末が受信可能なデータ量 (受信バッファサイズ) を送信元に通知します。

# 2. TCPアルゴリズム

2-1. TCPコネクションの確立

2-2. 到達確認

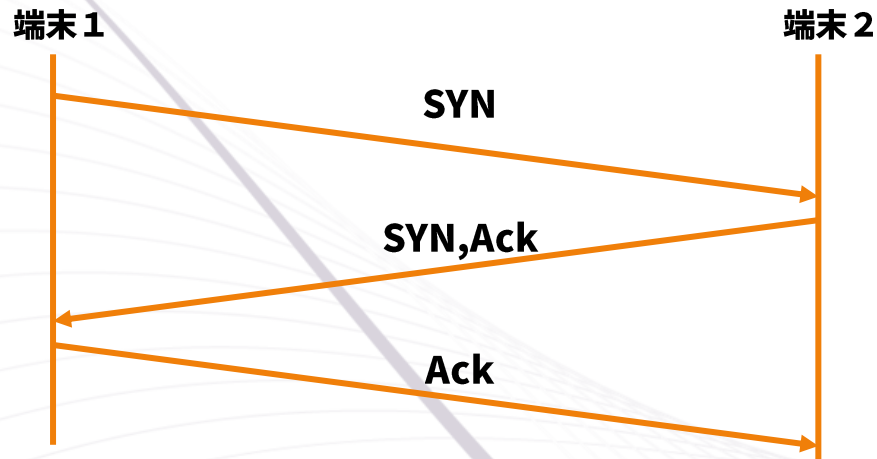
2-3. ウィンドウ制御

2-4. ウィンドウサイズの決定

2-5. 再送制御

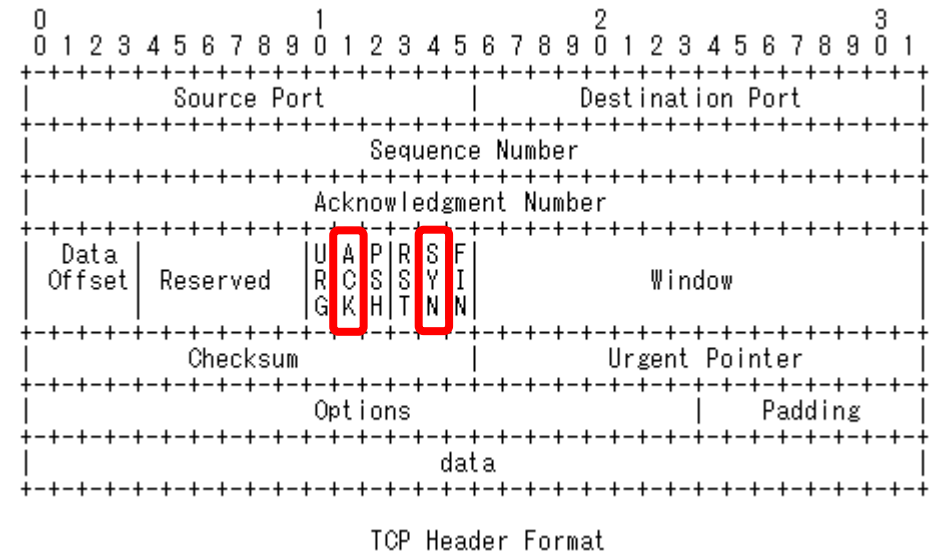
# 2-1. TCPコネクションの確立

## 3 Wayハンドシェイク



ここから以降で、データ通信が送信される

SelectiveAckやウィンドウサイズオプションなどのオプション機能の使用可否もこの確立時に決定されます。



# パケットキャプチャ (Wireshark)

http\_cap.pcapng

ファイル(F) 編集(E) 表示(V) 移動(G) キャプチャ(C) 分析(A) 統計(S) 電話(y) 無線(W) ツール(T) ヘルプ(H)

表示フィルタ... <Ctrl-/> を適用

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.7	157.7.107.210	TCP	66	53275 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.000364	192.168.1.7	157.7.107.210	TCP	66	59662 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	0.015081	157.7.107.210	192.168.1.7	TCP	66	80 → 53275 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1414 SACK_PERM=1 WS=128
4	0.015193	192.168.1.7	157.7.107.210	TCP	54	53275 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
5	0.016168	192.168.1.7	157.7.107.210	HTTP	614	GET / HTTP/1.1
6	0.016390	157.7.107.210	192.168.1.7	TCP	66	80 → 59662 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1414 SACK_PERM=1 WS=128
7	0.016462	192.168.1.7	157.7.107.210	TCP	54	59662 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8	0.031443	157.7.107.210	192.168.1.7	TCP	60	80 → 53275 [ACK] Seq=1 Ack=561 Win=30336 Len=0
9	0.542488	157.7.107.210	192.168.1.7	TCP	392	80 → 53275 [PSH, ACK] Seq=1 Ack=561 Win=30336 Len=338 [TCP segment of a reassembled PDU]
10	0.543525	157.7.107.210	192.168.1.7	TCP	2882	80 → 53275 [ACK] Seq=339 Ack=561 Win=30336 Len=2828 [TCP segment of a reassembled PDU]
11	0.543622	192.168.1.7	157.7.107.210	TCP	54	53275 → 80 [ACK] Seq=561 Ack=3167 Win=131328 Len=0
12	0.544684	157.7.107.210	192.168.1.7	TCP	8538	80 → 53275 [ACK] Seq=3167 Ack=561 Win=30336 Len=8484 [TCP segment of a reassembled PDU]

> Frame 5: 614 bytes on wire (4912 bits), 614 bytes captured (4912 bits) on interface \Device\NPF\_{EE298557-9F26-440F-870F-072F5825E194}, id 0

> Ethernet II, Src: Tp-LinkT\_09:d6:7d (28:ee:52:09:d6:7d), Dst: Mitsubis\_86:d6:65 (10:4b:46:86:d6:65)

> Internet Protocol Version 4, Src: 192.168.1.7, Dst: 157.7.107.210

> Transmission Control Protocol, Src Port: 53275, Dst Port: 80, Seq: 1, Ack: 1, Len: 560

> Hypertext Transfer Protocol

```
0000  10 4b 46 86 d6 65 28 ee 52 09 d6 7d 08 00 45 00  ·KF··e(· R·}··E·
0010  02 58 15 30 40 00 80 06 18 e7 c0 a8 01 07 9d 07  ·X·0@··· ······
0020  6b d2 d0 1b 00 50 38 57 b9 9e 16 b5 52 a1 50 18  k····P8W ····R·P·
0030  02 01 e5 e0 00 00 47 45 54 20 2f 20 48 54 54 50  ······GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 6d 61 6e 61  /1.1··Ho st: mana
0050  6b 61 6e 2e 6e 65 74 0d 0a 43 6f 6e 6e 65 63 74  kan.net· ·Connect
0060  69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d  ion: kee p-alive·
0070  0a 55 70 67 72 61 64 65 2d 49 6e 73 65 63 75 72  ·Upgrade ·Insecur
0080  65 2d 52 65 71 75 65 73 74 73 3a 20 31 0d 0a 55  e·Reques ts: 1·U
0090  73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c  ser·Agen t: Mozil
```

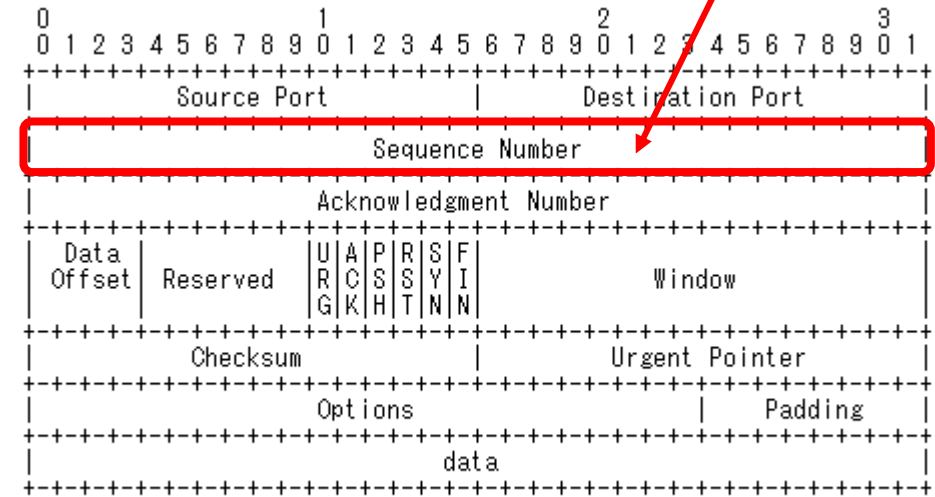
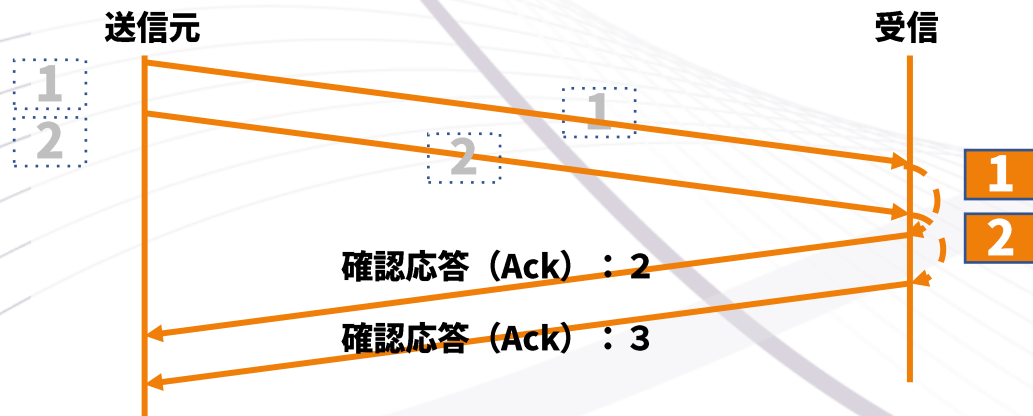
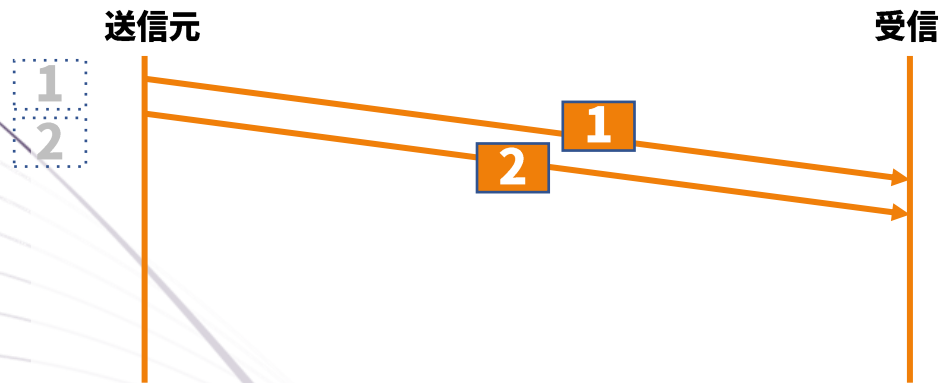
Transmission Control Protocol (tcp), 20 バイト

パケット数: 20 · 表示: 20 (100.0%)

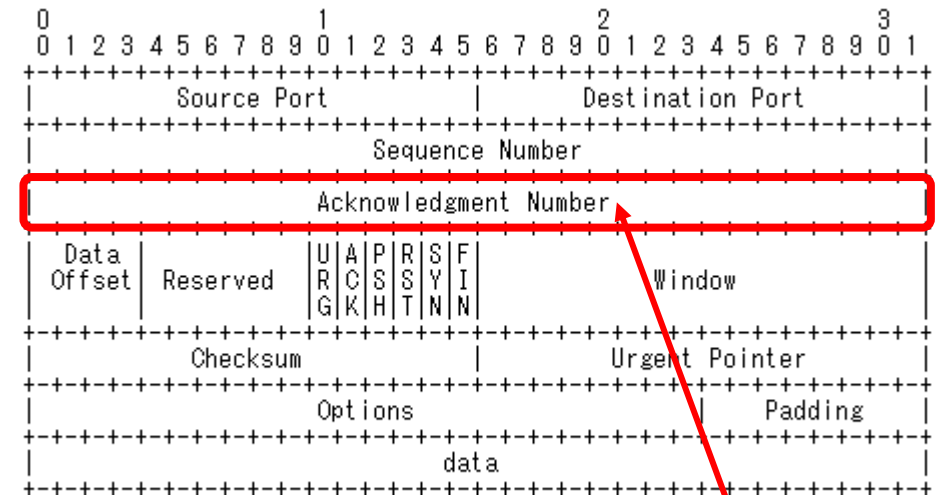
プロファイル: Default

# 2-2. 到達確認

送信データ番号



TCP Header Format



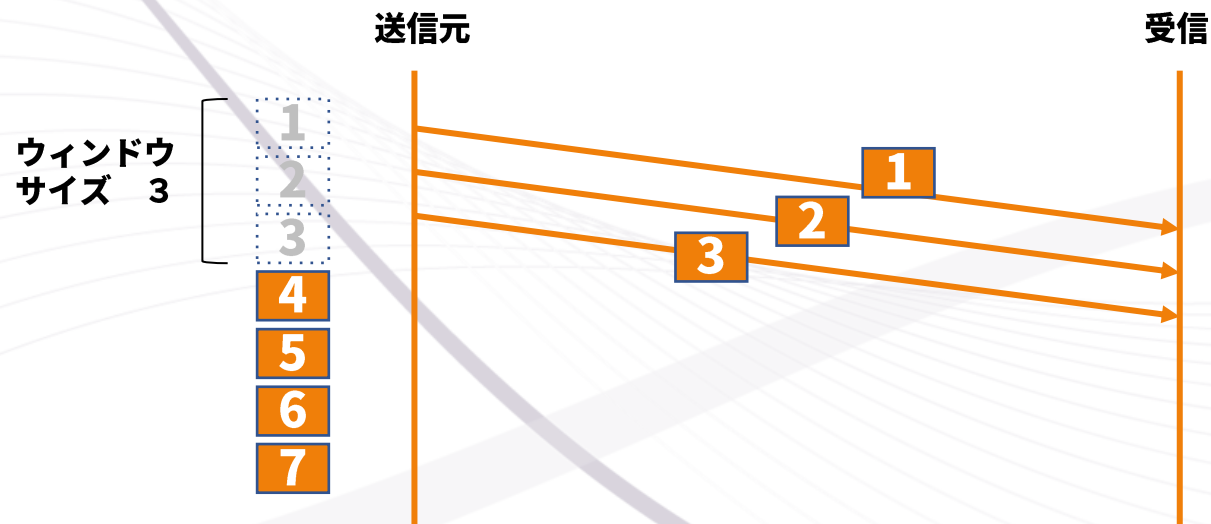
TCP Header Format

到達確認番号  
(次に受ける番号)

## 2-3. ウィンドウ制御

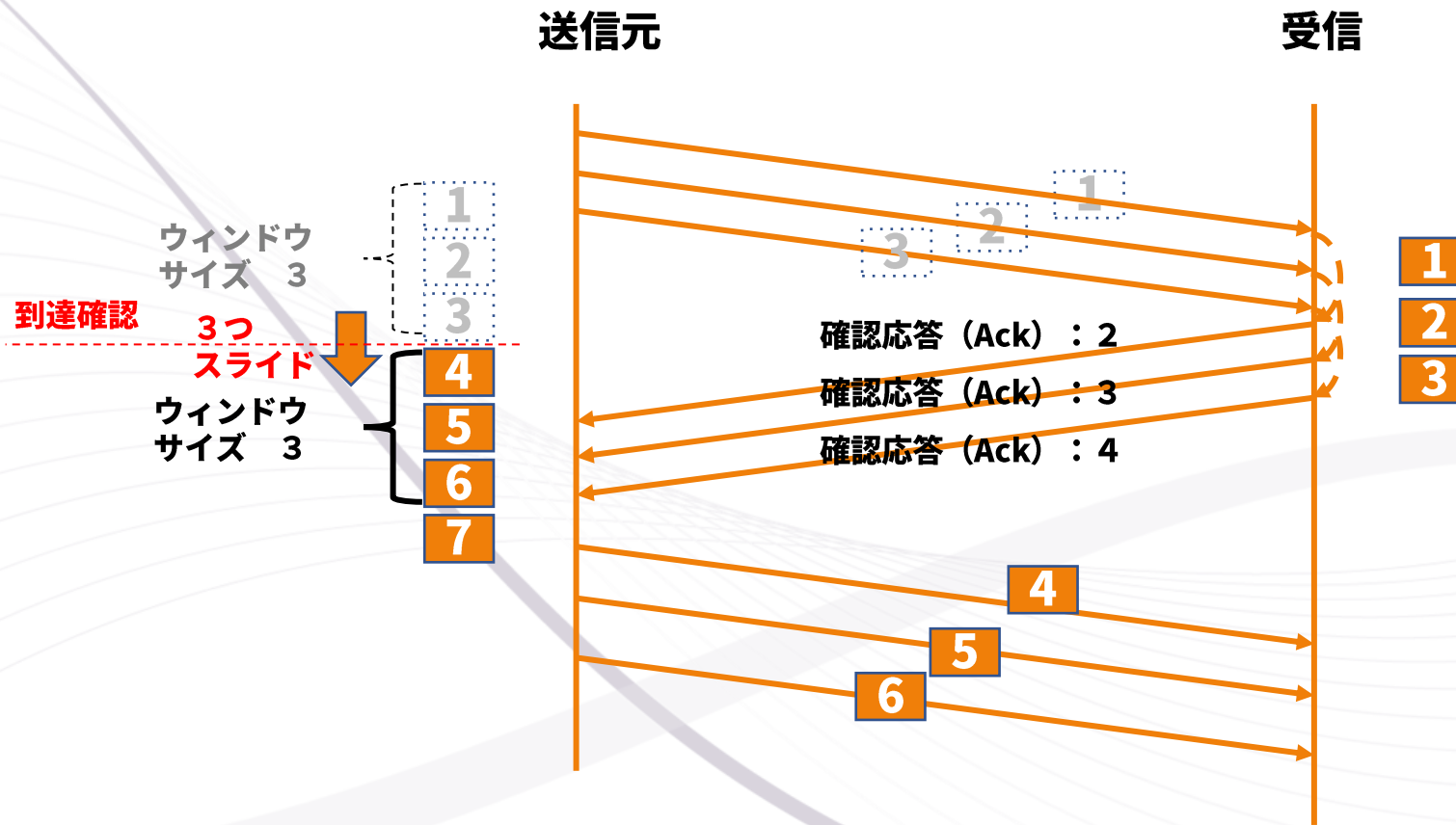
### ウィンドウとは

「到達確認できていないが  
送信することができるデータ量（バイト量）のこと」



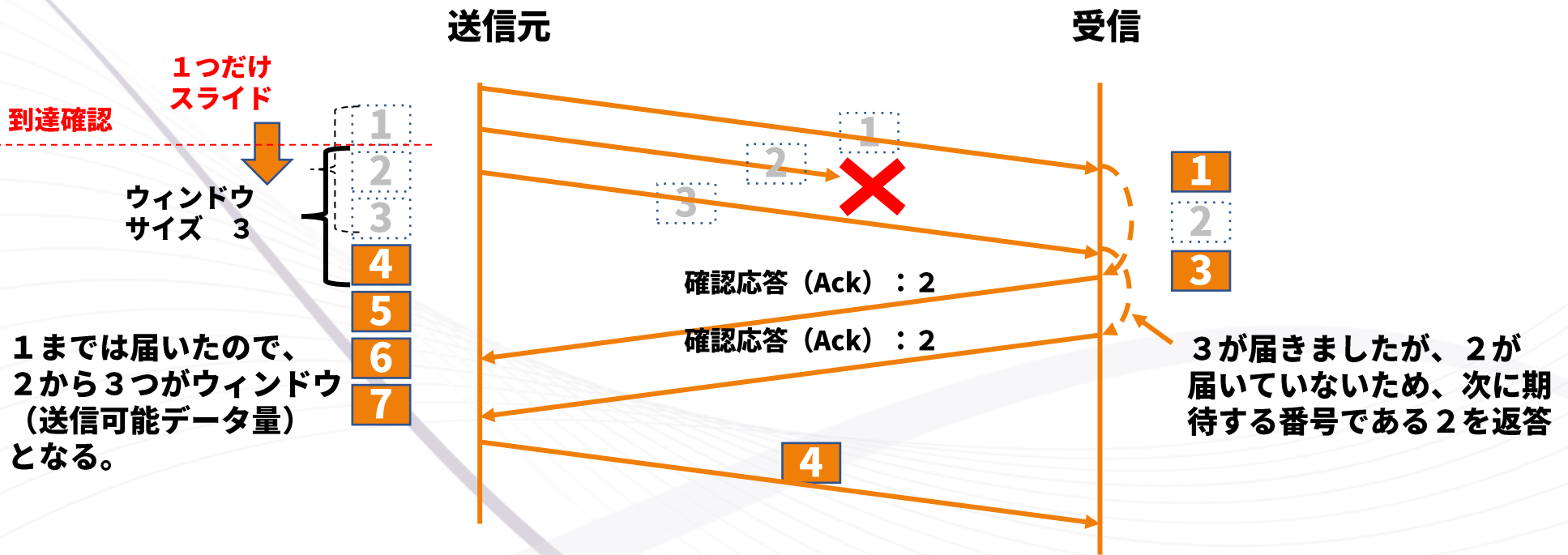
# 2-3. ウィンドウ制御

【全て正常に届いた場合】



# 2-3. ウィンドウ制御

【パケット 2 が損失した場合】





## 2-4. ウィンドウサイズの決定

ウィンドウサイズは送信できるサイズなので送信端末で決定されます。決定には以下の2つの内容の小さい値で決定されます。

### ① フロー制御

受信端末から通知されるバッファサイズ（ウィンドウサイズ）により制御

### ② 輻輳制御

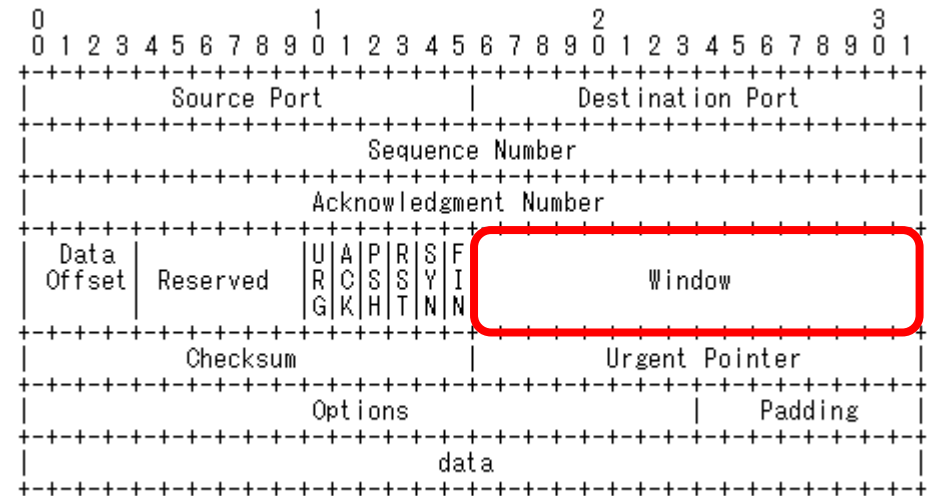
送信端末で計算されるウィンドウサイズにより制御

現在はパソコンの性能も向上しているため、受信端末のバッファサイズが影響することは少なく、送信側で計算される輻輳制御がウィンドウサイズになると考えて良いかと思います。

# 2-4-1. フロー制御

①に関しては、受信端末のバッファメモリが通知

- TCPパケットヘッダ「Window (16bit)」で通知
- 最大サイズ16bitで表示できる65535Byte (64Kbyte)
- 近年は最大サイズを拡張できるウィンドウサイズオプションの利用が一般的



TCP Header Format

## 2-4-2. 輻輳制御（輻輳ウィンドウ）

②輻輳制御は輻輳ウィンドウによって制御される。

輻輳ウィンドウは送信元端末の内部で保持し計算される値

パケットヘッダ内には情報は出てこない。

パソコンの高性能化により①フロー制御の受信端末バッファサイズよりも②輻輳制御の送信元端末で計算される輻輳ウィンドウサイズがウィンドウサイズの支配的。

（ただし、オプションなしでは16Bitの64Kbyteが最大値となり支配的であるため、ウィンドウサイズオプションにより拡張されるのが一般的です。）

# 2-4-2. 輻輳制御（輻輳ウィンドウ）

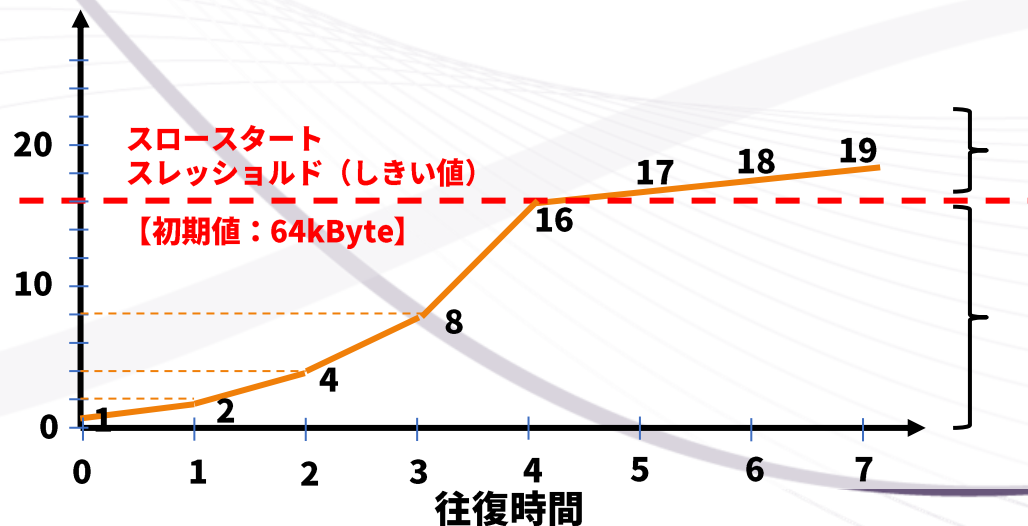
輻輳ウィンドウ(cwnd: Congestion Window) : Byteサイズ（説明はパケット数）

輻輳ウィンドウサイズはMSS（MaximumSegmentSize）単位で計算されます。

$$\begin{aligned} \text{MSS} &= \text{Ether最大パケットサイズ} - \text{IPヘッダ長} - \text{TCPヘッダ長} \\ &= 1500 \text{ Byte} - 20 \text{ Byte} - 20 \text{ Byte (オプション無)} \\ &= 1460 \text{ Byte} \end{aligned}$$

【スロースタート】と【輻輳回避】の2つのフェーズがあり、輻輳ウィンドウサイズの算出方法が異なる。

輻輳ウィンドウ  
(cwnd)  
サイズ



【輻輳回避】

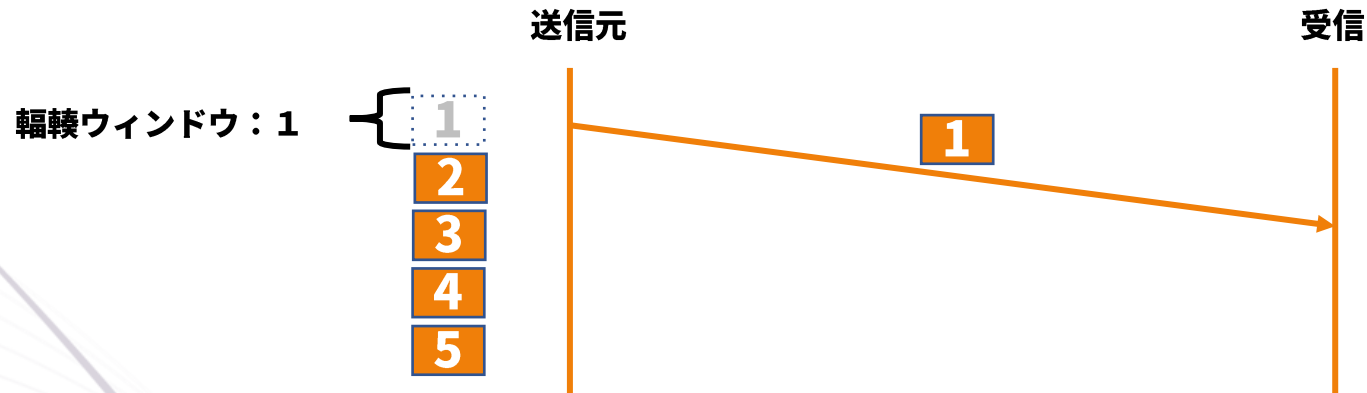
往復時間毎に1MSSサイズ（線形的）増加  
【計算式】  
 $\text{cwnd} = \text{cwnd} + \text{MSS}/\text{cwnd}$

【スロースタート】

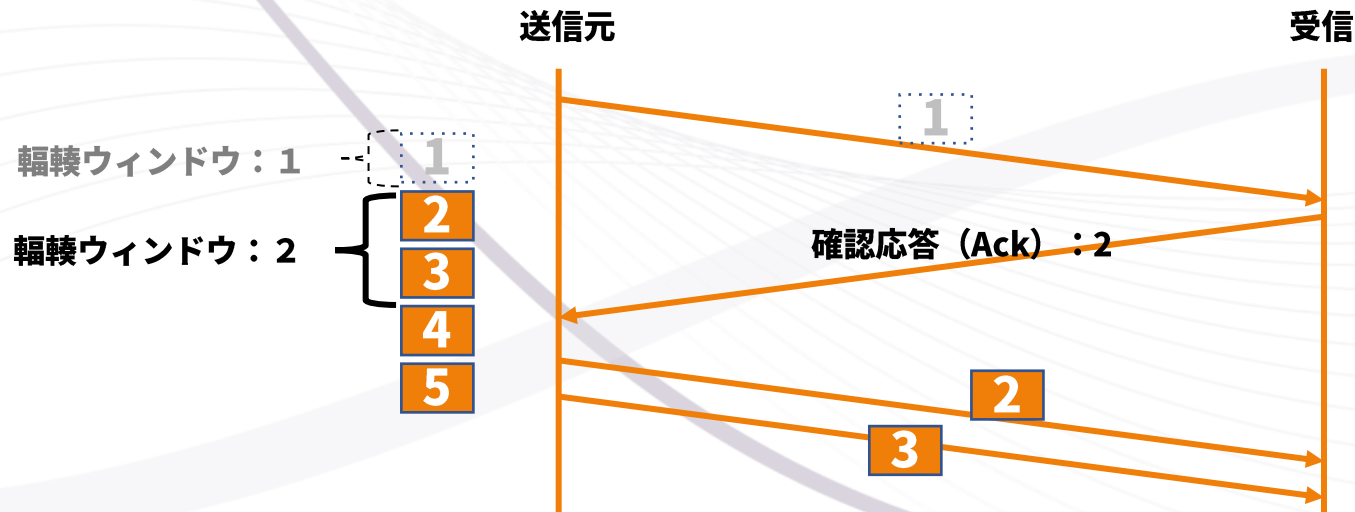
往復時間毎に指数関数的に増加  
【計算式】  
 $\text{cwnd} = \text{cwnd} + \text{MSS}$

# 2-4-2-1. スロースタート

【コネクション確立時】



【確認応答1受信時】

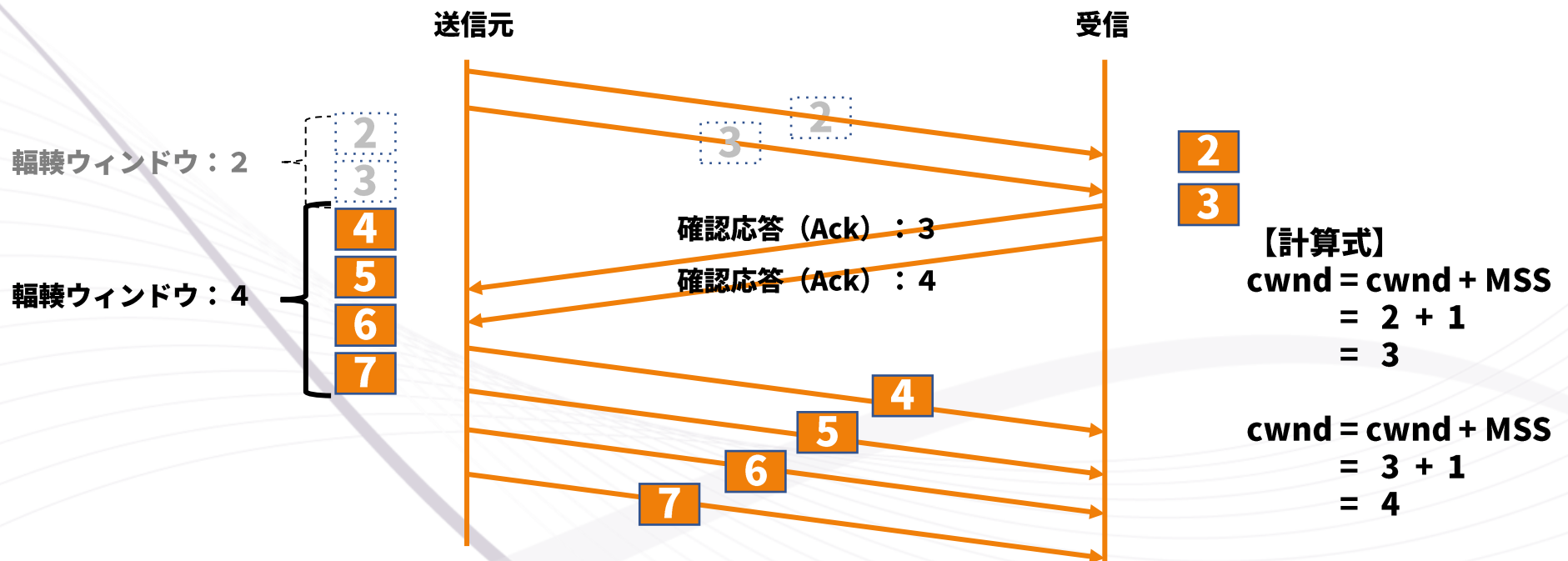


【計算式】

$$\begin{aligned} \text{cwnd} &= \text{cwnd} + \text{MSS} \\ &= 1 + 1 \\ &= 2 \end{aligned}$$

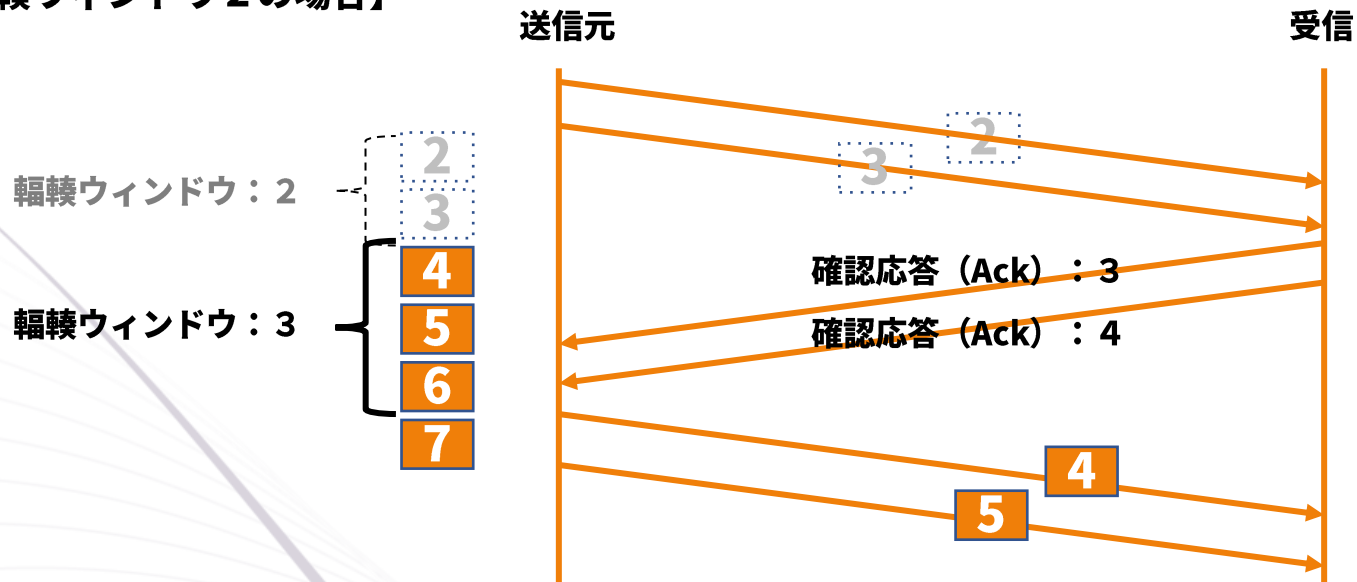
# 2-4-2-1. スロースタート

【確認応答3まで受信時】



# 2-4-2-2. 輻輳回避

【輻輳ウィンドウ2の場合】

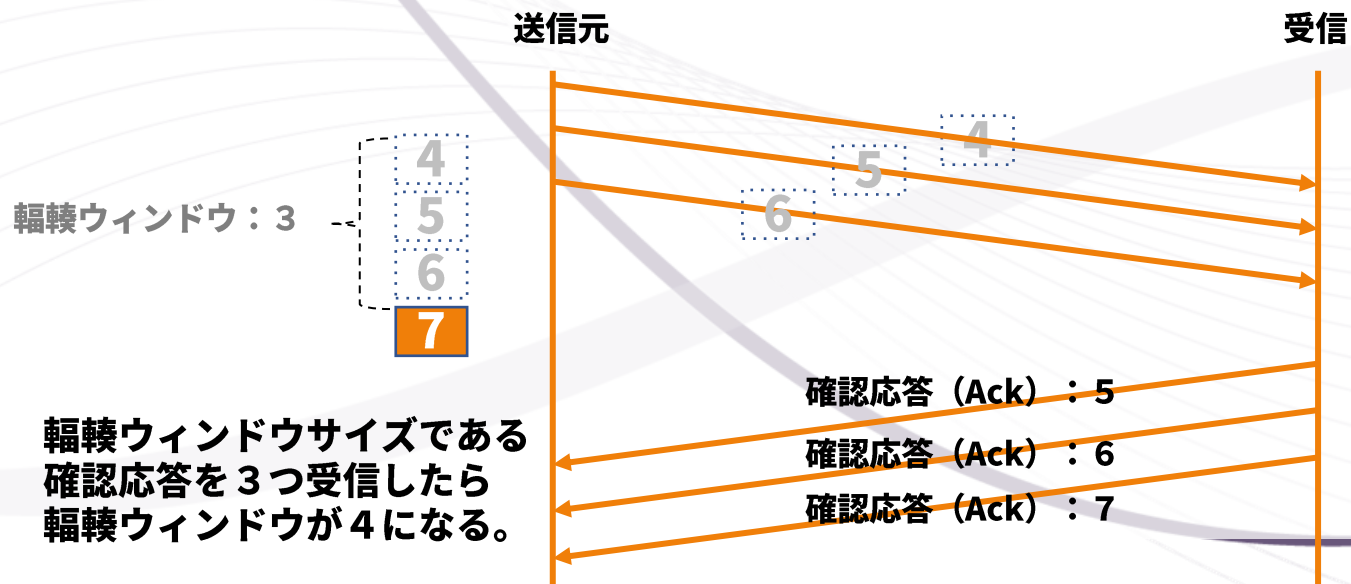


【計算式】

$$\begin{aligned} \text{cwnd} &= \text{cwnd} + \text{MSS}/\text{cwnd} \\ &= 2 + 1/2 \\ &= 2 \frac{1}{2} \end{aligned}$$

$$\begin{aligned} \text{cwnd} &= \text{cwnd} + \text{MSS}/\text{cwnd} \\ &= 2 \frac{1}{2} + 1/2 \\ &= 3 \end{aligned}$$

【輻輳ウィンドウ3の場合】



【計算式】

$$\begin{aligned} \text{cwnd} &= \text{cwnd} + \text{MSS}/\text{cwnd} \\ &= 3 + 1/3 \\ &= 3 \frac{1}{3} \end{aligned}$$

$$\begin{aligned} \text{cwnd} &= \text{cwnd} + \text{MSS}/\text{cwnd} \\ &= 3 \frac{1}{3} + 1/3 \\ &= 3 \frac{2}{3} \end{aligned}$$

$$\begin{aligned} \text{cwnd} &= \text{cwnd} + \text{MSS}/\text{cwnd} \\ &= 3 \frac{2}{3} + 1/3 \\ &= 4 \end{aligned}$$

## 2-5. 再送制御

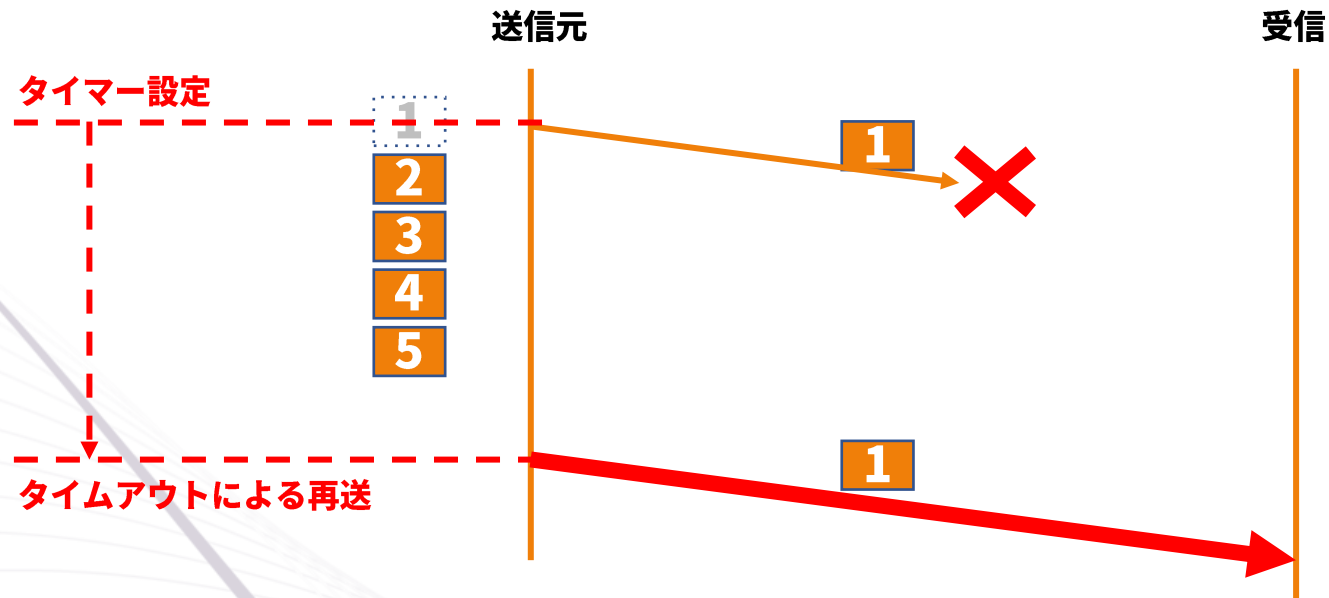
インターネットでは、ネットワーク内でパケットが損失することが前提で構築されています。そのため、TCPでは確実にデータを届けるため再送制御機能を有しています。再送機能については以下の2点の機能を実装しています。

① タイムアウトによる再送

② 重複Ackによる再送



## 2-5-1. タイムアウトによる再送

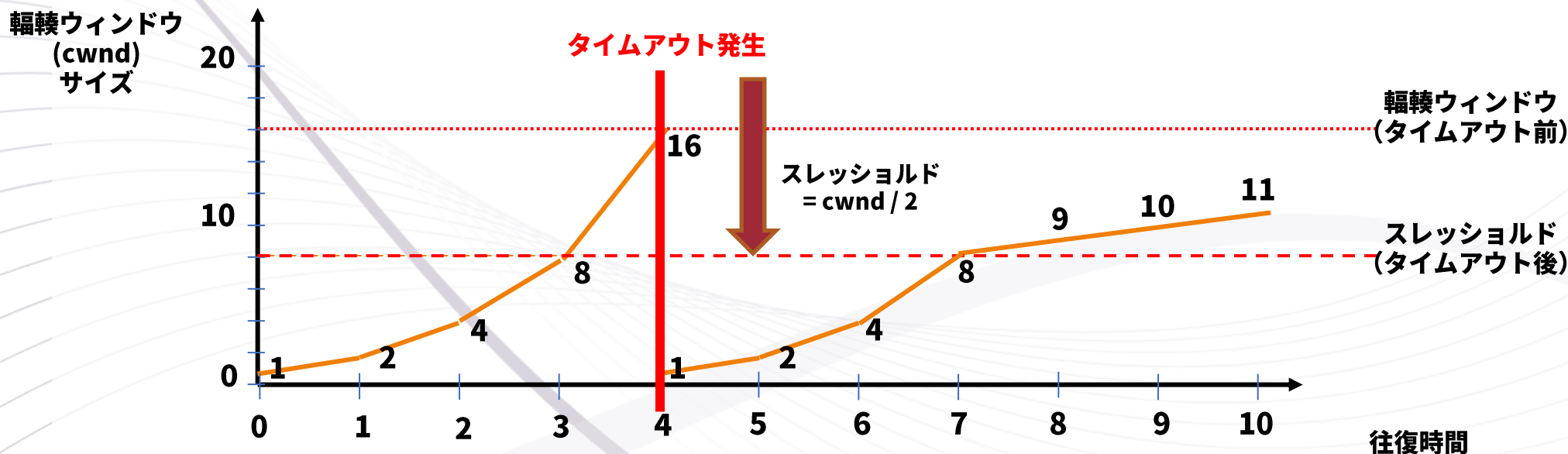


タイマー時間は初期値は決まっていますが、実際のパケットの応答時間から随時更新されます。

# 2-5-1. タイムアウトによる再送

タイムアウトによる再送が発生した場合には、以下のウィンドウ制御が行われ、輻輳ウィンドウサイズの変動を図にすると以下ようになります。

- ①輻輳ウィンドウ(cwnd) = 1 MSSサイズ
- ②スロースタートスレッシュヨルド =  $cwnd / 2$

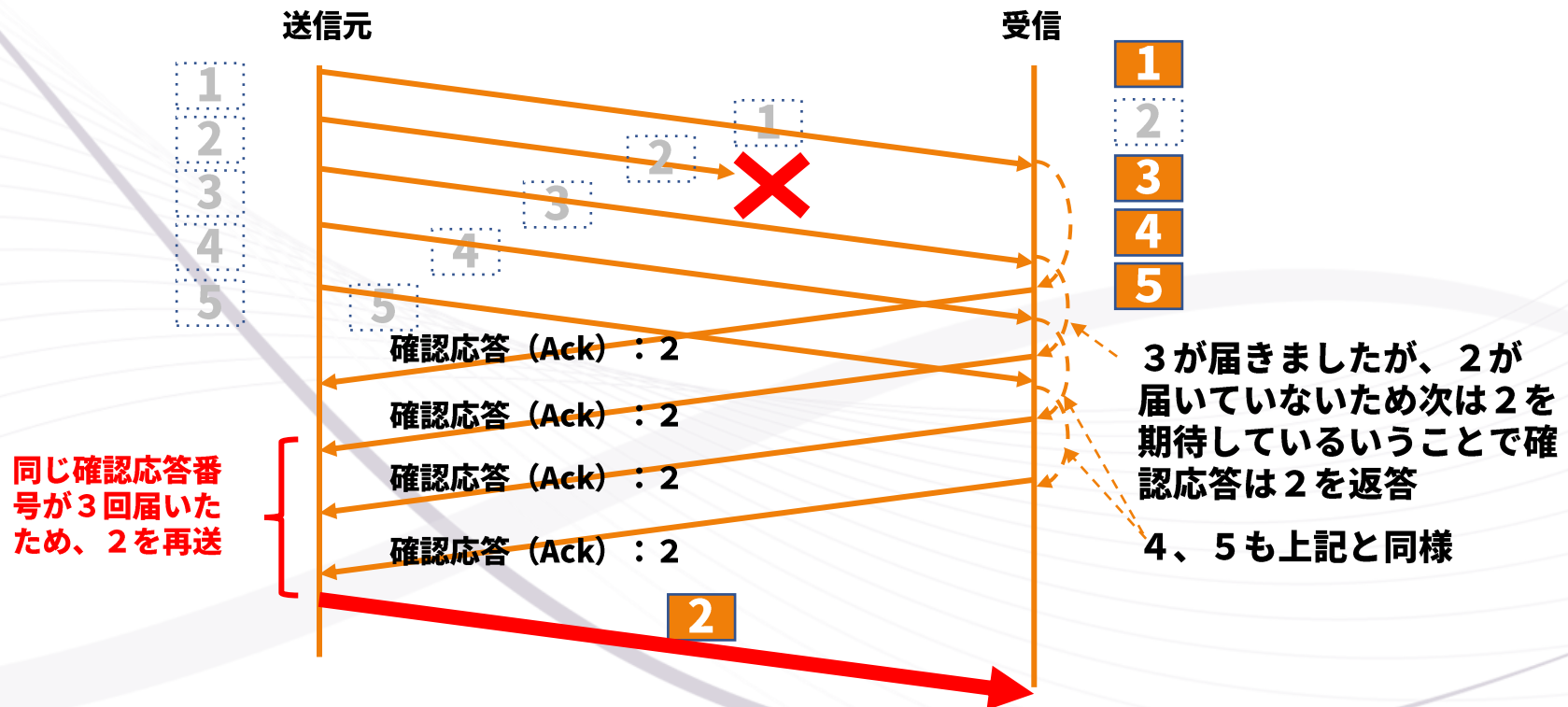


タイムアウトするということはネットワークは非常に混雑していると判断します。そのため、無駄なパケット損失を招かないように慎重に送信して少しずつウィンドウを広げていく必要があります。

## 2-5-2. 重複Ackによる再送

「高速再転送アルゴリズム」  
重複Ackによる再送は、以前と同じシーケンス番号が3回届いたときに再送。

【仮にパケット2が損失した場合】



# 2-5-2. 重複Ackによる再送

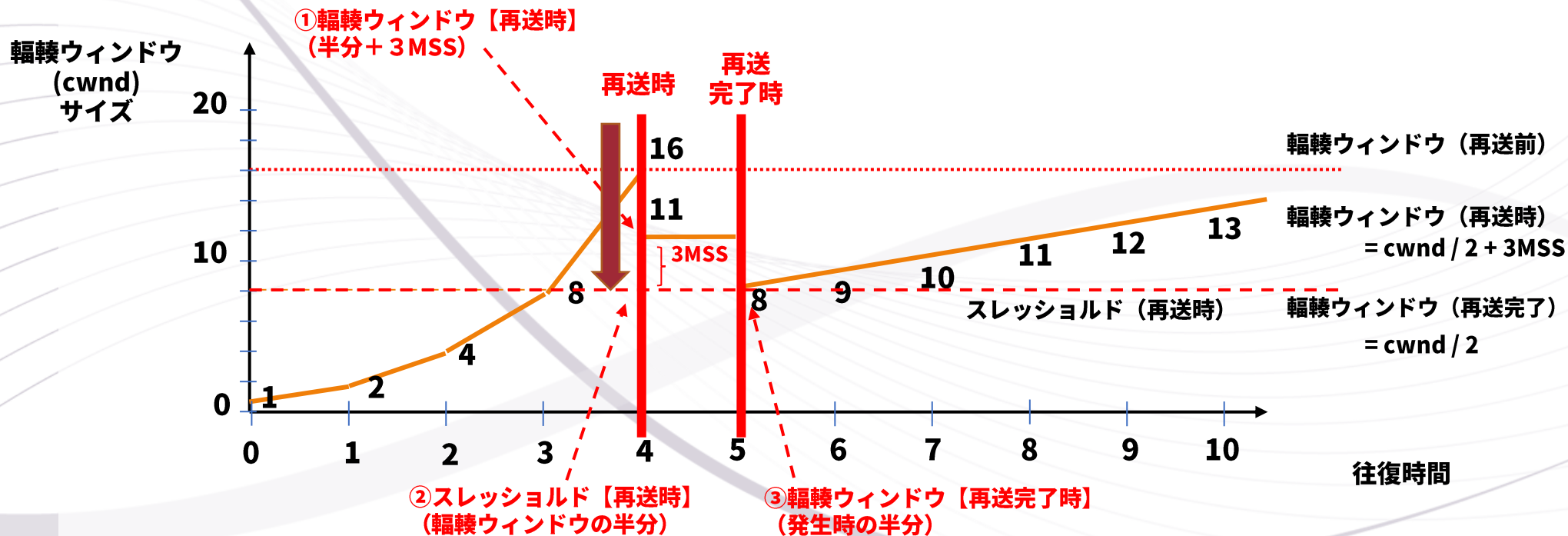
高速リカバリ (FastRecovery) 機能：輻輳ウィンドウを最初の1MSSサイズではなく、以下の計算を実装し、再送によるデータ伝送を効率的に行っています。

【再送発生時】

- ①輻輳ウィンドウ(cwnd) =  $cwnd / 2 + 3MSS$
- ②スロースタートスレッシュヨルド =  $cwnd / 2$

【再送完了時】

- ③輻輳ウィンドウ(cwnd) = スロースタートスレッシュヨルド



# 3. まとめ【TCPの主な伝送制御】

## ●到達確認

## ●ウィンドウ制御

- ・フロー制御
- ・輻輳制御
  - スロースタート
  - 輻輳回避

## ●再送制御

- ・タイムアウトによる再送 ← 【混雑大】
- ・重複ACKによる再送 ← 【混雑小】  
(高速再転送アルゴリズム)

**チャンネル登録、ご意見、ご要望の程**

**よろしくお願ひします。**